Running long and complex processes with PostGIS

Vincent Picavet

FOSS4G 2010 - Barcelona
Oslandia, who's that?
Oslandia

Young French SME specialised in Open Source GIS

PostGIS experts: Vincent Picavet & Olivier Courtin

- Mainly Focuses on:
  - **Spatial Databases** (PostGIS, SpatiaLite)
  - OGC, ISO, INSPIRE **Standards** and **SDI architecture**
  - **Complex analysis**: Routing, Network and Graphs Solutions

Oslandia's ecosystem:
Oslandia's Technologies

3D  GDAL  GEOS

GRASS  GraphServer  INSPIRE  MapServer

OGC  PgRouting  PostGIS

PostgreSQL  Spatialite  TinyOWS

TileCache  PyWPS  QGIS
Running long and complex processes with PostGIS

Vincent Picavet, Wednesday - 12h00 – Sala 6

PostGIS meets the third dimension

Olivier Courtin, Wednesday - 12h30 – Sala 6

State of the Art of FOSS4G for Topology and Network Analysis

Vincent Picavet, Thursday - 14h30 – Sala 5

Breakout Session: Spatial Databases

Code Sprint on Friday: PostGIS
What you'll see and do next

- **Step 1**: *Use case* presentation
- **Step 2**: Special use *characteristics*
- **Step 3**: *Issues* and *solutions*
- **Step 4**: *Conclusion*
- **Step 5**: *Perspectives*

- **Step 6**: Stay here for Olivier's presentation
- **Step 7**: Run for lunch
Step 1 : Use case
Use case

- Road network data (TA)
- + Custom client data linked to the network
- Initial network data imported in 2004
- Parallel evolution during 4 years
  - Client modified road network data
  - TA modified road network data
- No ID stability on TA data
  → data de-synchronization
Use case

Desynchronization

Red: custom data
Background: road network (rasterized)
Left 2004 right 2008
Goal

- **Re-synch** custom data with up-to-date network
  - **Graph pairing**
    = match networks streets, nodes, road sections
  - Re-link or rebuild custom data on new network
  - Have a full road network data update process
- **Automate** this process
  - Enable fully automated and regular data update
Our process

1. Load data
2. Graph pairing modules (nodes, streets, sections)
   - Semantic, topological and geometrical subprocesses
3. Export output data
### Facts & numbers

- **Our data set**
  - 70% of French population (~40M)
  - 50 Tables
  - 10M rows
  - 150Go at end of process
  - 30K SQL and plpgsql lines
  - 3000 queries, 6000 Python lines

- **Our dev team**
  - 3 Mapinfo users and 1 PostGIS expert
Results

- 2004 → 2008:
  - 70% road sections pairing
  - *93% custom data pairing*

- 2008 → 2009:
  - 99% road sections pairing
  - *99.95% custom data pairing*

- Less difference between networks
- Custom data have been cleaned
Step 2 : Characteristics
Use case characteristics

- «ELT»: Extract, Load, Transform
- PostgreSQL + PostGIS + external tools
- Big volumes
- **Long, heavy and complex** computation process
  - Global production time ~ 20 days
  - Pairing: 5 days
- **Long SQL transactions**
Step 3: Issues and solutions
Issues and solutions

#1 – Hardware and server configuration
#2 – Testing
#3 – Monitoring
#4 – Dealing with corner cases
#5 – Splitting process
#6 – Stability
#7 – Optimization
#8 – Process improvement

⇒ Almost all of this is linked to the way you **design your process**.
#1 – Hardware and configuration

- Adapted hardware is essential
  - Buy **RAM**
  - Buy more **RAM**
  - Buy more **RAM**
  - Buy disks
  - Buy faster disks

- Server configuration is **hard**
  - System monitoring
  - Depends on the process
  - Dynamic configuration
    - Fine-tune according to query plan
  - Needs experience
  - Needs testing

And use it!
#2 – Testing

- **Testing for correctness**
  - Ok on sample for development
  - Corner case problems on full data

- **Testing for performance**
  - Meaningless on samples
  - Very long on real data

- **Solutions**
  - Split process
  - «Unit test» modules
  - Guess and oversize everything
#3 – Monitoring, validating - MVCC

- **MVCC** = Multi-View Concurrency Control
  - → concurrent access on data
  - → Transactions isolated until committed
  - → No easy way to access a running transaction

- Use smaller transactions
- Sequence monitoring: sequences live out of MVCC
  - `nextval('myseq')` in query
  - `currval('myseq')` gets progression
#3 – Monitoring and validating

- **System** monitoring
  - Memory, disk access
  - Shows process stability and steps

- **Post-process** monitoring and validation
  - Log analysis
  - Validation processes on result tables
  - Statistics on result tables

- **Intra-process** monitoring and validation
  - Split process
#4 – Corner cases - issue

- Computations with geometry is

  **not an exact science**

  <= Data error & imprecision
  <= Floating point models limits
  <= Robustness of algorithms
  <= Error propagation
#4 – Corner cases - issue

99.999999% success
1M rows

\[ \{ 1 \text{ Geometry computation error} \}
\]

transaction fails!

- Every additional «9» costs a lot more than precedent
  - Performance-wise, code complexity-wise
- Success rate drops with computation complexity
  \[ \leq \text{Error propagation} \]

→ **Impossible to predict** all corner cases
#4 – Corner cases - Actual Solutions

- **Split** process in chunks

- Preprocess and simplify data
  - Snap to grid (= reduce input precision)
  - Simplify

- Catch errors to ignore them
  - Using exception catching in plpgsql
    - Not precise enough (catch all)
    - Less stability
#4 – Corner cases - Potential solutions

- Finely handle **errors**
  - Specific exceptions
  - Discuss use cases to decide returning NULL or error

- Change floating point models
  - Enable **custom FP models** (in JTS and GEOS, not PostGIS)
  - Dynamic floating point precision model
  - Exact computation (costs a lot)

- More robust algorithms
#5 – Split your process

- Split computations
- Split data
- Not possible in plpgsql
  - <= no nested transaction
- Needs a process driver
  - Python is our driver
- Enables
  - Intra-process operations
    Backup, validate, stats, monitor...)
  - partial computation & diff updates
  - // computation
#5 – Split your process

Python Script vs plpgsql Transaction

Input data split

Agglomerate Result data

Python Script
#6 – Stability

- Memory management in PG is smart
  - Memory allocated and freed per transaction context
  - PostGIS uses it, not GEOS
- Longer transactions
- Some GEOS memory leaks
- Catching geometric errors

\[
\text{increase instability}
\]

- Use recent PostgreSQL release
#7 – Optimizing

- **Indexes**
  - Necessary for geometric operations
  - Must be finely tuned
  - Drop, modify, recreate (automated in plpgsql)

- **Constraints**
  - Same: drop, modify, recreate
  - Or replace by validation steps

- **Maintenance**
  - Vacuum vs autovacuum

- **Quit plpgsql**
  - PostgreSQL C modules are fun! – and efficient
#8 – Process improvement

- Less geometry computation
- More topology and attribute-based processes
- Base computation on input data
  - Less computation errors
  - Less error propagation
  - Use original cleaned data
- Use PostGIS mainly:
  - in data preparation
  - geometry rebuilding at the end
Step 4 : Conclusion
**So what?**

- It works!
- Good results at the end

- Ease of use for PostgreSQL/PostGIS newbie developers
  - With expert assistance on problematic points

- Designing the process workflow carefully and thoroughly is the key
Step 5 : Perspectives
What more then?

- PostgreSQL improvement
  - HOT standby => parallel work
  - Nested transaction support ?
  - Better autovacuum
- In our case
  - Horizontal process split effort
  - Parallel processing
  - Differential work
- NoSQL «DB» ?
  - Map/Reduce system
That's all folks!

Want to know more?
Ask now or write to:

Vincent Picavet
vincent.picavet@oslandia.com

www.oslandia.com